C# 프로그래밍

- 3 주차 (1강)

SEJONG UNIVERSITY

비트 연산자

SEJONG UNIVERSITY

비트 연산자 종류

연산자	연산자 이름	설명	지원 형식
<<	왼쪽 시프트	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 왼쪽으로 이동	첫 번째 피연산자는 int, unit, long, ulong 이고, 두번째 피연산자는 int 형식
>>	오른쪽 시프트	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 오른쪽으로 이동	첫 번째 피연산자는 int, unit, long, ulong 이고, 두번째 피연산자는 int 형식
&	논리곱 (AND)	두 피연산자의 비트 논리곱을 수행	정수 계열 형식과 bool 형식
|	논리합 (OR)	두 피연산자의 비트 논리합을 수행	정수 계열 형식과 bool 형식
^	배타적 논리합(XOR)	두 피연산자의 비트 배타적 논리합을 수행	정수 계열 형식과 bool 형식
~	보수 (NOT)	피연산자의 비트를 0은 1로, 1은 0으로 반전	int, uint, long, ulong 형식

사용 예제

```
static void Main(string[] args)
{
    int a = 9, b = 10;
    Console.WriteLine($"a << 1 : {a << 1}");
    Console.WriteLine($"b >> 2 : {b >> 2}");
    Console.WriteLine($"a & b : {a & b}");
    Console.WriteLine($"a | b : {a | b}");
    Console.WriteLine($"a ^ b: {a ^ b}");
    Console.WriteLine($"~a : {~a}");
}
```

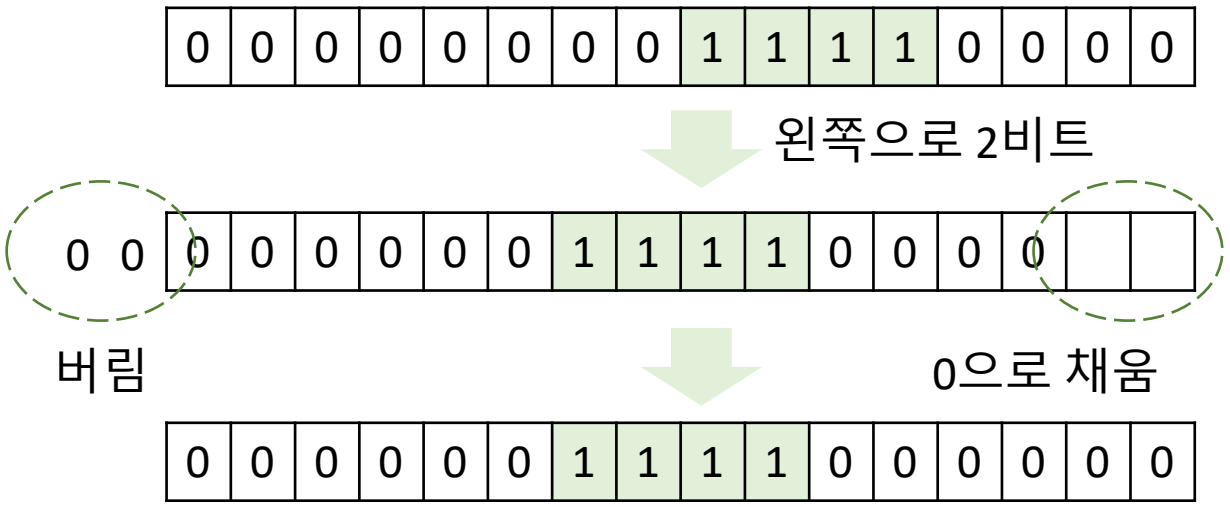
출력 결과:

```
a << 1 : 18
b >> 2 : 2
a & b : 8
a | b : 11
a ^ b: 3
~a : -10
```

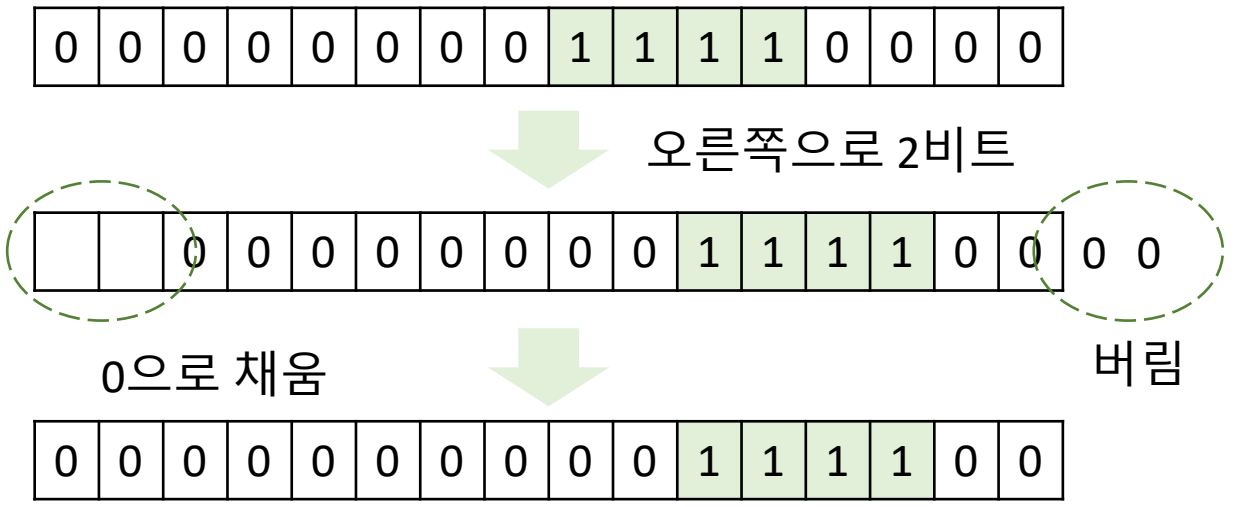
비트 연산자: 시프트 연산자

- 비트를 왼쪽(<<) 또는 오른쪽 (>>)으로 이동시키는 연산자
- 예를 들어, 10진수 240을 16비트로 표현 후 왼쪽/오른쪽 시프트
  - ✓ 시프트 연산자 지원형식은 32비트 이상이지만, 설명을 위해 16비트 사용

240 << 2 = 960



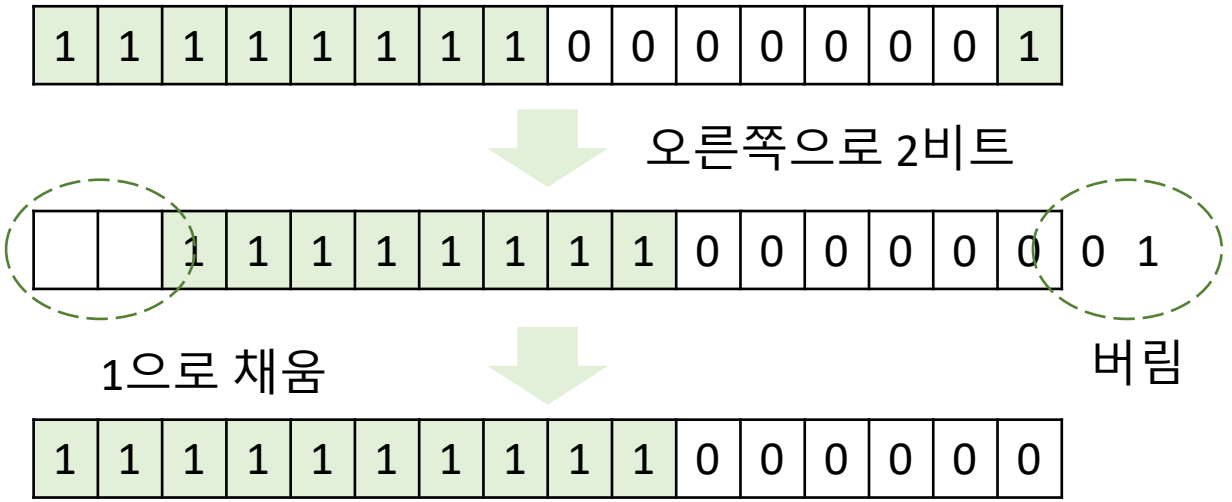
240 >> 2 = 60



음수에 대한 시프트 연산자

- 비트 이동 후 빈자리에 0이 아닌 1로 채움
- 예를 들어, -255를 비트로 표현하면 1111 1111 0000 0001

-255 >> 2 = -64



시프트 연산자 활용

- 고속의 곱셈과 나눗셈을 구현
  - ✓ 원본 데이터를 a, 옮긴 비트 수를 b라고 할 때,
  - ✓ $a \ll b = a \times 2^b$ 이고, $a \gg b = a \div 2^b$
  - ✓ 예를 들어, 240의 왼쪽 2비트 시프트 결과는 960 이고, 오른쪽 2비트 시프트 결과는 60

- 작은 단위로 쪼개진 데이터를 큰 데이터 형식으로 재조립
  - ✓ byte 형식의 데이터들을 하나의 int 또는 long 형식으로 표현
  - ✓ 논리곱(&), 논리합(|) 연산자와 함께 사용

시프트 연산자 예제 코드

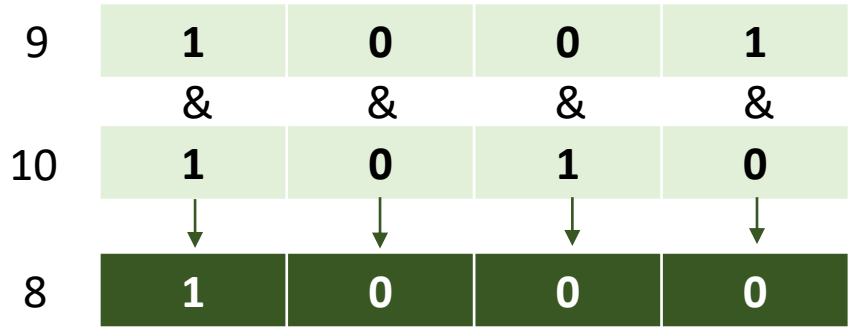
```
static void Main(string[] args)
{
    int a = 1;      // Testing <<...
    Console.WriteLine("a     : {0:D5} (0x{0:X8})", a);
    Console.WriteLine("a << 1: {0:D5} (0x{0:X8})", a << 1);
    Console.WriteLine("a << 2: {0:D5} (0x{0:X8})", a << 2);
    Console.WriteLine("a << 5: {0:D5} (0x{0:X8})", a << 5);

    int b = 255;    // Testing >>...
    Console.WriteLine("b     : {0:D5} (0x{0:X8})", b);
    Console.WriteLine("b >> 1: {0:D5} (0x{0:X8})", b >> 1);
    Console.WriteLine("b >> 2: {0:D5} (0x{0:X8})", b >> 2);
    Console.WriteLine("b >> 5: {0:D5} (0x{0:X8})", b >> 5);

    int c = -255;   // Testing negative number >>...
    Console.WriteLine("c     : {0:D5} (0x{0:X8})", c);
    Console.WriteLine("c >> 1: {0:D5} (0x{0:X8})", c >> 1);
    Console.WriteLine("c >> 2: {0:D5} (0x{0:X8})", c >> 2);
    Console.WriteLine("c >> 5: {0:D5} (0x{0:X8})", c >> 5);
}
```

출력 결과:

```
a     : 00001 (0x00000001)
a << 1: 00002 (0x00000002)
a << 2: 00004 (0x00000004)
a << 5: 00032 (0x00000020)
b     : 00255 (0x000000FF)
b >> 1: 00127 (0x0000007F)
b >> 2: 00063 (0x0000003F)
b >> 5: 00007 (0x00000007)
c     : -00255 (0xFFFFFF01)
c >> 1: -00128 (0xFFFFFF80)
c >> 2: -00064 (0xFFFFFFC0)
c >> 5: -00008 (0xFFFFFFF8)
```

비트 논리 연산자: 논리곱, 논리합

- bool 형식 외에 정수 계열 형식의 피연산자에 대해도 사용
- 논리곱(&) 연산자
  - ✓ 두 비트 모두 1(참)이어야 결과도 1(참)



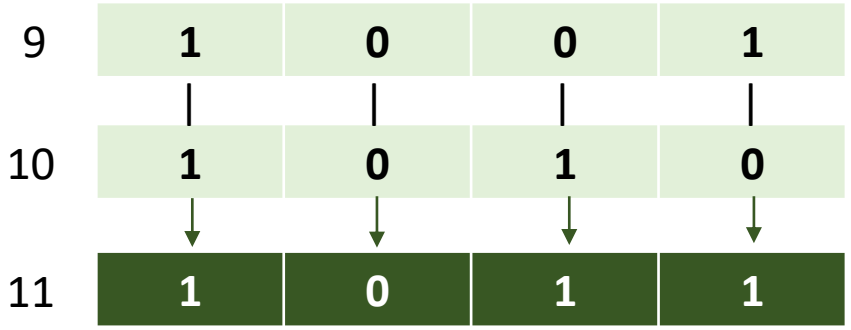
C# 코드

```
int result = 9 & 10;     // result는 8
```

- 논리합(|) 연산자
  - ✓ 두 비트 중 하나라도 1(참)이면 결과도 1(참)

C# 코드

```
int result = 9 | 10;     // result는 11
```

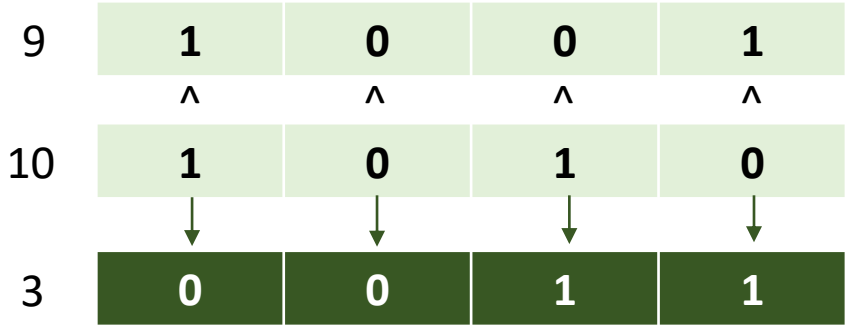


비트 논리 연산자: 배타적 논리합

- 배타적 논리합(^) 연산자
  - ✓ 두 비트 진리 값이 서로 달라야 1(참)

C# 코드

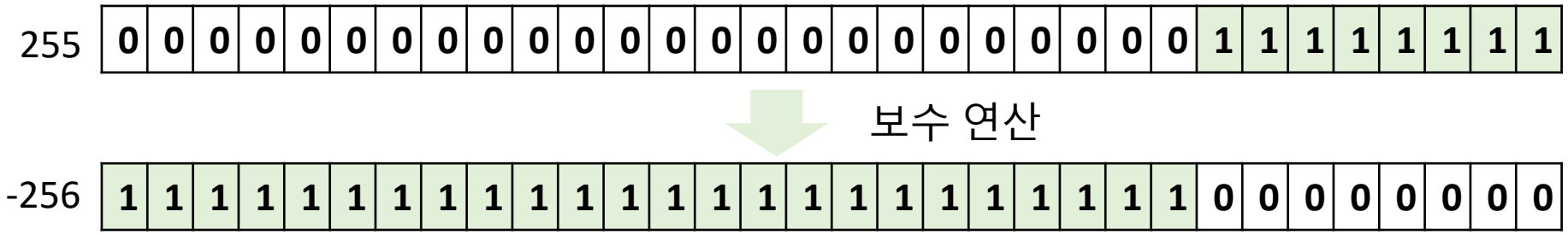
```
int result = 9 ^ 10;     // result는 3
```



- 보수(~) 연산자
  - ✓ 단항 연산자로 비트를 0에서 1로, 1에서 0으로 반전

```
int a = 255;
int result = ~a; // result는 -256
```

C# 코드



할당 연산자

- 복합 할당 연산자 (+=, -=, *= 등등)
  - ✓ 왼쪽 피연산자와 오른쪽 피연산자를 '='의 왼쪽 연산자로 가공 후,
  - ✓ 다시 왼쪽 피연산자에 할당

연산자	연산자 이름	설명
=	할당 연산자	오른쪽 피연산자를 왼쪽 피연산자에 할당
+=	덧셈 할당 연산자	a += b;는 a = a + b;와 같음
-=	뺄셈 할당 연산자	a -= b;는 a = a - b;와 같음
*=	곱셈 할당 연산자	a *= b;는 a = a * b;와 같음
/=	나눗셈 할당 연산자	a /= b;는 a = a / b;와 같음
%=	나머지 할당 연산자	a %= b;는 a = a % b;와 같음
&=	논리곱 할당 연산자	a &= b;는 a = a & b;와 같음
|=	논리합 할당 연산자	a |= b;는 a = a | b;와 같음
^=	배타적 논리합 할당 연산자	a ^= b;는 a = a ^ b;와 같음
<<=	왼쪽 시프트 할당 연산자	a <<= b;는 a = a << b;와 같음
>>=	오른쪽 시프트 할당 연산자	a >>= b;는 a = a >> b;와 같음

할당 연산자 예제 코드

```
static void Main(string[] args)
{
    int a = 100;
    Console.WriteLine($"a += 90 : {a += 90}");
    Console.WriteLine($"a -= 80 : {a -= 80}");
    Console.WriteLine($"a *= 70 : {a *= 70}");
    Console.WriteLine($"a /= 60 : {a /= 60}");
    Console.WriteLine($"a %= 50 : {a %= 50}");
    Console.WriteLine($"a &= 40 : {a &= 40}");
    Console.WriteLine($"a |= 30 : {a |= 30}");
    Console.WriteLine($"a ^= 20 : {a ^= 20}");
    Console.WriteLine($"a <<= 10 : {a <<= 10}");
    Console.WriteLine($"a >>= 10 : {a >>= 1}");
}
```

출력 결과:

```
a += 90 : 190
a -= 80 : 110
a *= 70 : 7700
a /= 60 : 128
a %= 50 : 28
a &= 40 : 8
a |= 30 : 30
a ^= 20 : 10
a <<= 10 : 10240
a >>= 10 : 5120
```

코드 흐름 제어

- 분기문 (if, switch)
- 반복문
- 점프문

SEJONG UNIVERSITY

분기문: if 문

- 프로그램의 흐름을 조건에 따라 여러 갈래로 나누는 구문
- if ~ else if ~ else 조건문
  - ✓ if 문에서 사용하는 조건식은 true 또는 false의 값을 가지는 bool 형식

```
if ( 조건식 )
{
    // if 문의 조건식이 참인 경우 실행
}
else if ( 조건식 )
{
    // if문의 조건식이 거짓이고, elseif 문의 조건식이 참인 경우 실행
}
else
{
    // if문과 elseif 문의 조건식이 모두 거짓인 경우 실행
}
```

if 문 예제 코드

```
static void Main(string[] args)
{
    Console.WriteLine("숫자를 입력하세요. : ");

    string input = Console.ReadLine();
    int number = Int32.Parse(input);

    if (number < 0)
        Console.WriteLine("음수");
    else if (number > 0)
        Console.WriteLine("양수");
    else
        Console.WriteLine("0");

    if (number % 2 == 0)
        Console.WriteLine("짝수");
    else
        Console.WriteLine("홀수");
}
```

```
출력 결과:
>IfElse
숫자를 입력하세요. :
33
양수
홀수

>IfElse
숫자를 입력하세요. :
-12
음수
짝수
```

if 문 중첩 예제 코드

```
static void Main(string[] args)
{
    Console.Write("숫자를 입력하세요. : ");

    string input = Console.ReadLine();
    int number = Int32.Parse(input);

    if (number > 0)
    {
        if (number % 2 == 0)
            Console.WriteLine("0보다 큰 짝수");
        else
            Console.WriteLine("0보다 큰 홀수");
    }
    else
    {
        Console.WriteLine("0보다 작거나 같은 수.");
    }
}
```

출력 결과:

```
>IfIf
숫자를 입력하세요. : 11
0보다 큰 홀수

>IfIf
숫자를 입력하세요. : 4
0보다 큰 짝수

>IfIf
숫자를 입력하세요. : -10
0보다 작거나 같은 수.
```

분기문: switch 문

- 조건식의 결과가 가질 수 있는 다양한 경우를 평가
  - ✓ C언어와 달리 조건식에서 정수 뿐 아니라 문자열 형식 지원

```
switch ( 조건식 )
{
    case 상수1:
        // 조건식이 상수1 이면, 실행할 코드
        break;
    case 상수2:
        // 조건식이 상수2 이면, 실행할 코드
        break;
    case 상수N:
        // 조건식이 상수N 이면, 실행할 코드
        break;
    default:
        // 모든 case가 거짓인 경우, 실행할 코드
        break;
}
```

- break 문
  - ✓ 프로그램 흐름을 멈추고 현재 실행 중인 코드의 바깥으로 실행 위치를 옮김
  - ✓ goto 또는 return과 같은 점프문을 대신하여 사용 가능

switch 문 예제 코드

```
static void Main(string[] args)
{
    Console.Write("요일을 입력하세요.(일,월,화,수,목,금,토) : ");
    string day = Console.ReadLine();

    switch (day)
    {
        case "일":
            Console.WriteLine("Sunday");
            break;
        case "월":
            Console.WriteLine("Monday");
            break;
        case "화":
            Console.WriteLine("Tuesday");
            break;
        case "수":
            Console.WriteLine("Wednesday");
            break;
        case "목":
            Console.WriteLine("Thursday");
            break;
        case "금":
            Console.WriteLine("Friday");
            break;
        case "토":
            Console.WriteLine("Saturday");
            break;
        default:
            Console.WriteLine($"{day}는 확인불가.");
            break;
    }
}
```

출력 결과:

```
>Switch
요일을
입력하세요.(일,월,화,수,목,금,토) : 화
Tuesday
```

데이터형식 따라 분기하는 switch 문

- C# 7.0 부터 switch 문에 데이터 형식을 조건으로 사용
  - ✓ case 절에 데이터 형식 옆에 식별자 반드시 포함

```
object obj = 123;        // 컴파일러에 의해 123 리터럴을 int형으로 인식
switch ( obj )
{
    case int i:
        // obj에 담겨 있는 데이터 형식이 int형 일 때, 실행할 코드
        break;
    case float f:
        // obj에 담겨 있는 데이터 형식이 float 형 일 때, 실행할 코드
        break;
    default:
        // obj에 담겨 있는 데이터 형식이 int/float이 아닐 때, 실행할 코드
        break;
}
```

데이터형식 따라 분기하는 예제 코드

```
static void Main(string[] args)
{
    object obj = null;

    string s = Console.ReadLine();
    if (int.TryParse(s, out int out_i))
        obj = out_i;
    else if (float.TryParse(s, out float out_f))
        obj = out_f;
    else
        obj = s;

    switch (obj)
    {
        case int i:
            Console.WriteLine($"{i}는 int 형식.");
            break;
        case float f:
            Console.WriteLine($"{f}는 float 형식.");
            break;
        default:
            Console.WriteLine($"{obj}는 모르는 형식.");
            break;
    }
}
```

- TryParse ()
  - ✓ 모든 숫자형식에서 포함하는 메서드
  - ✓ 문자열을 숫자형식 데이터로 변환
  - ✓ 변환 성공 시 true 반환, 실패 시 false 반환
  - ✓ out 키워드를 통한 출력 전용 매개변수 사용

```
출력 결과:
>Switch2
32
32는 int 형식.

>Switch2
123.45
123.45는 float 형식.
```

추가 조건 검사를 위한 when 절

```
switch (obj)
{
    case int i:
        Console.WriteLine($"{i}는 int 형식.");
        break;
    case float f when f >= 0: // obj가 float 형식이고 0보다 크거나 같은 경우
        Console.WriteLine($"{f}는 양의 float 형식.");
        break;
    case float f:             // obj가 float 형식이고 0보다 작은 경우
        Console.WriteLine($"{f}는 음의 float 형식.");
        break;
    default:
        Console.WriteLine($"{obj}는 모르는 형식.");
        break;
}
```

switch 식

- switch 문과 switch 식과의 차이
  - ✓ switch 문은 어떤 작업에 분기가 필요 할 때 사용하고, switch 식은 분기를 거쳐 결과 값을 내놓아야 할 때 사용

```
bool repeated = true;
switch (score)
{
    case 90:
        grade = "A";
        break;
    case 80 when repeated == true:
        grade = "B+";
        break;
    case 80:
        grade = "B";
        break;
    case 70:
        grade = "C";
        break;
    default:
        grade = "F";
        break;
}
```

- switch 문에서 switch 식으로 변환
  - ✓ 조건식을 switch 키워드 앞으로 위치
  - ✓ case 키워드와 ":" 대신에 "=>" 사용
  - ✓ "break;" 대신에 콤마(,) 사용
  - ✓ default 키워드 대신에 "_" 사용
  - ✓ 세미콜론 ";"으로 종료

switch 식으로 바꾼 코드

```
bool repeated = true;
string grade = score switch
{
    90 => "A",
    80 when repeated == true => "B+",
    80 => "B",
    70 => "C",
    _ => "F"
};
```

switch 식 예제 코드

```
static void Main(string[] args)
{
    Console.WriteLine("점수를 입력하세요");
    int score = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("재수강인가요? (y/n)");
    string line = Console.ReadLine();
    bool repeated = line == "y" ? true : false;

    string grade = (int)(Math.Truncate(score/10.0)*10) switch
    {
        90 when repeated == true => "B+",
        90 => "A",
        80 => "B",
        70 => "C",
        60 => "D",
        _ => "F"
    };
    Console.WriteLine($"학점 : {grade}");
}
```

출력 결과:

```
>SwitchExp2
점수를 입력하세요
92
재수강인가요? (y/n)
y
학점 : B+
```

코드 흐름 제어

- 분기문
- 반복문
(while, do while, for, foreach)
- 점프 문

while 문, do while 문

- while 문
  - ✓ 조건식이 참인 동안 코드를 반복 실행

사용 형식

```
while ( 조건식 )
{
    // 반복 실행할 코드
}
```

- do while 문
  - ✓ 처음 한번은 코드를 실행 후, 조건식이 참인 동안 코드를 반복 실행

사용 형식

```
do
{
    // 반복 실행할 코드
}
while (조건식);
```

while 문 사용 예제

```
int a = 10;
while (a > 0)
{
    Console.WriteLine(a);
    a -= 2;
}
```

do while 사용 예제

```
int a = 10;
do
{
    Console.WriteLine(a);
    a -= 2;
}
while (a > 0);
```

while 문, do while 문 예제 코드

```
static void Main(string[] args)
{
    int i = 10;
    while (i > 0)
    {
        Console.WriteLine("a) i : {0}", i--);
    }

    do
    {
        Console.WriteLine("b) i : {0}", i--);
    }
    while (i > 0);
}
```

출력 결과:

```
a) i : 10
a) i : 9
a) i : 8
a) i : 7
a) i : 6
a) i : 5
a) i : 4
a) i : 3
a) i : 2
a) i : 1
b) i : 0
```

for 문

- 조건식이 참인 동안 코드를 반복 실행
- while 문 보다 반복을 더 정교하게 제어
- 초기화식
  - ✓ 반복을 실행하기 전에 최초에 한번 실행하는 코드로써 보통 변수 초기화 수행
- 조건식
  - ✓ 반복을 계속 수행할지를 결정
- 반복식
  - ✓ 반복이 끝날 때마다 실행, 주로 조건식에서 사용되는 변수 값 조정

사용 형식

```
for (초기화식; 조건식; 반복식)
{
    // 반복 실행할 코드
}
```

for 문 사용 예제

```
for (int i=0; i<5; i++)
{
    Console.WriteLine(i);
}
```

for 문 예제 코드

```
static void Main(string[] args)
{
    for (int i=0; i<5; i++)
    {
        for (int j = 0; j <= i; j++)
        {
            Console.Write("*");
        }
        Console.WriteLine();
    }
}
```

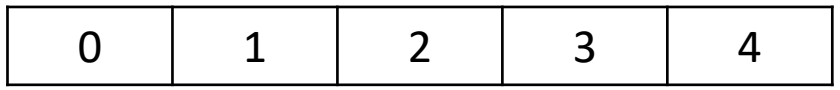
출력 결과:

```
*
**
***
****
*****
```

foreach 문

- 배열 (또는 컬렉션)을 순회하면 각 데이터 요소에 차례로 접근
- 배열 (또는 컬렉션) 끝에 도달하면 자동으로 반복이 종료
- "in" 키워드와 함께 사용
  - ✓ 배열의 각 요소를 순회하면서 in 키워드 앞에 있는 변수에 할당
- 배열
  - ✓ 여러 개의 데이터를 담을 수 있는 코드 요소

예: `int[] arr = new int[] { 0, 1, 2, 3, 4 };`



사용 형식

```
foreach (데이터형식 변수명 in 배열(또는 컬렉션))
{
    // 반복 실행할 코드
}
```

foreach 문 사용 예제

```
static void Main(string[] args)
{
    int[] arr = new int[] { 0, 1, 2, 3, 4 };
    foreach (int a in arr)
    {
        Console.WriteLine(a);
    }
}
```

코드 흐름 제어

- 분기문
- 반복문
- 점프문
(break, continue, goto, return, throw)

break 문

- 현재 실행 중인 반복문이나 switch 문의 실행을 중단하고자 할 때 사용

```
static void Main(string[] args)
{
    while(true)
    {
        Console.Write("계속할까요?(예/아니오) : ");
        string answer = Console.ReadLine();

        if (answer == "아니오")
            break;
    }
}
```

출력 결과:

```
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 예
계속할까요?(예/아니오) : 아니오
```

continue 문

- 반복문에서 한 회 코드블록 실행을 건너 뛰어 반복을 계속 수행
  - ✓ continue 문을 만나면 코드블록 안에서 그 아래 코드 자동 실행 취소
  - ✓ 가독성이 좋은 특징을 가짐

```
static void Main(string[] args)
{
    for (int i =0; i<10; i++)
    {
        if (i % 2 == 0)
            continue;

        Console.WriteLine($"{i} : 홀수");
    }
}
```

```
출력 결과:
1 : 홀수
3 : 홀수
5 : 홀수
7 : 홀수
9 : 홀수
```

goto 문

- 코드 안에 레이블을 정의 goto 문을 만나면 바로 레이블로 이동
  - ✓ 중첩된 반복문을 지정한 레이블 위치로 단숨에 빠져나올 수 있는 장점
  - ✓ 가독성이 안 좋게 만드는 단점

사용 형식

```
goto 레이블;

레이블:
    // 이어지는 코드
```

사용 예제

```
{
    Console.WriteLine("1");

    goto JUMP;

    Console.WriteLine( “2");
    Console.WriteLine( “3");

    JUMP:
    Console.WriteLine( “4");
}
```

Thank you!